IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Spyker et al.**	§	Group Art Unit: **2137**
Serial No.: **09/595,287**	§	Examiner: **Norris, Tremayne M.**
Filed: **June 15, 2000**	§	Attorney Docket No.:**RSW9-2000-0042-US1**
For: **Apparatus and Method for Ensuring Data Integrity of Unauthenticated Code**	§	

36736
PATENT TRADEMARK OFFICE
CUSTOMER NUMBER

Certificate of Mailing Under 37 C.F.R. § 1.8(a)
I hereby certify this correspondence is being deposited with the United States Postal Service as First Class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on August 5, 2004.

By: Michele Morrow
Michele Morrow

TRANSMITTAL DOCUMENT

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:
ENCLOSED HEREWITH:

- Appellant's Brief (in triplicate) (37 C.F.R. 1.192); and
- Our return postcard.

A fee of $330.00 is required for filing an Appellant's Brief. Please charge this fee to IBM Corporation Deposit Account No. 09-0461. No additional fees are believed to be necessary. If, however, any additional fees are required, I authorize the Commissioner to charge these fees which may be required to IBM Corporation Deposit Account No. 09-0461. No extension of time is believed to be necessary. If, however, an extension of time is required, the extension is requested, and I authorize the Commissioner to charge any fees for this extension to IBM Corporation Deposit Account No. 09-0461.

Respectfully submitted,

Duke W. Yee
Registration No. 34,285
YEE & ASSOCIATES, P.C.
P.O. Box 802333
Dallas, Texas 75380
(972) 367-2001
ATTORNEY FOR APPLICANTS

AF/2137
120
#9
10/3

Docket No. RSW9-2000-0042-US1

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Spyker et al.**	§	
	§	Group Art Unit: **2137**
Serial No. **09/595,287**	§	
	§	Examiner: **Norris, Tremayne M.**
Filed: **June 15, 2000**	§	
	§	
For: **Apparatus and Method for Ensuring Data Integrity of Unauthenticated Code**	§ § §	

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

ATTENTION: Board of Patent Appeals and Interferences

<u>Certificate of Mailing Under 37 C.F.R. § 1.8(a)</u>
I hereby certify this correspondence is being deposited with the United States Postal Service as First Class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on August 5, 2004.

By: Michele Morrow
Michele Morrow

APPELLANT'S BRIEF (37 C.F.R. 1.192)

This brief is in furtherance of the Notice of Appeal, filed in this case on June 8, 2004.

The fees required under § 1.17(c), and any required petition for extension of time for filing this brief and fees therefore, are dealt with in the accompanying TRANSMITTAL OF APPEAL BRIEF.

This brief is transmitted in triplicate. (37 C.F.R. 1.192(a))

REAL PARTIES IN INTEREST

The real party in interest in this appeal is the following party: International Business Machines Corporation.

RELATED APPEALS AND INTERFERENCES

With respect to other appeals or interference's that will directly affect, or be directly affected by, or have a bearing on the Board's decision in the pending appeal, there are no such appeals or interference's.

STATUS OF CLAIMS

A. TOTAL NUMBER OF CLAIMS IN APPLICATION

Claims in the application are: 1-29.

B. STATUS OF ALL THE CLAIMS IN APPLICATION

1. Claims canceled: NONE
2. Claims withdrawn from consideration but not canceled: NONE
3. Claims pending: 1-29
4. Claims allowed: NONE
5. Claims rejected: 1-29

C. CLAIMS ON APPEAL

The claims on appeal are: 1-29.

STATUS OF AMENDMENTS

There are no amendments after final rejection.

SUMMARY OF INVENTION

The present invention provides an apparatus and method for ensuring data integrity of unauthenticated code. (Specification, page 12, lines 27-29) In particular, the present invention is directed to an apparatus and method for authenticating native code based on hash value information obtained in automatically authenticated code, e.g., platform independent code. (Specification, page 18, lines 6-17) With the present invention, a hash value of unauthenticated code is embedded in associated automatically authenticated code. (Specification, page 20, lines 11-13) When the automatically authenticated code is downloaded and executed, the automatically authenticated code may require that the unauthenticated code also be downloaded for proper execution of the automatically authenticated code on a particular client device. (Specification, page 20, lines 21-31) The unauthenticated code can be downloaded and its integrity verified by generating a hash value of the unauthenticated code and comparing the generated hash value to a hash value embedded in the automatically authenticated code. (Specification, page 20, line 32 to page 21, line 10) Since the hash value of the unauthenticated code is embedded in authenticated code, and the authenticated code must have passed its authentication check or it would not have been executed, the embedded hash value can be trusted not to have been changed and can safely be used to determine whether the unauthenticated code has changed. (Specification, page 23, lines 9-19)

ISSUES

The issues on appeal are whether:

1) claims 1, 2, 4-6, 11, 12, 14-16, 21, 22 and 24-26 are anticipated under 35 U.S.C. § 102(b) based on McManis (U.S. Patent Number 5,692,047);

2) claims 3, 10, 13, 20 and 23 are unpatentable under 35 U.S.C. § 103(a) over McManis (U.S. Patent Number 5,692,047); and

3) claims 7-9, 17-19 and 27-29 are unpatentable under 35 U.S.C. § 103(a) over McManis (U.S. Patent Number 5,692,047) in view of Cordery et al. (U.S. Patent Number 6,480,831).

GROUPING OF CLAIMS

The claims stand or fall together as a single group.

ARGUMENT

The Final Office Action rejects claims 1, 2, 4-6, 11, 12, 14-16, 21, 22 and 24-26 under 35 U.S.C. § 102(b) as being allegedly anticipated by McManis (U.S. Patent Number 5,692,047). Additionally, the Final Office Action rejects claims 3, 10, 13, 20 and 23 under 35 U.S.C. § 103(a) as allegedly being unpatentable over McManis (U.S. Patent Number 5,692,047). Further, the Final Office Action rejects claims 7-9, 17-19 and 27-29 under 35 U.S.C. § 103(a) as allegedly being unpatentable over McManis (U.S. Patent Number 5,692,047) in view of Cordery et al. (U.S. Patent Number 6,480,831).

Appellants respectfully submits that, contrary to the allegations made in the Final Office Action, McManis does not, in fact, teach or suggest the feature of receiving automatically authenticated code, the automatically authenticated code including an embedded first hash value of the unauthenticated code, as recited in independent claims 1, 11 and 21.

McManis is directed towards a system and method in which a program interpreter for programs, whose integrity is verifiable, includes a facility for using non-verifiable programs from trusted sources. In addition, McManis provides a system and method for refusing to execute other non-verifiable programs. The McManis system includes a program executer that executes verifiable architecture neutral programs and a class loader that prohibits the loading and execution of non-verifiable programs unless the non-verifiable program resides in a trusted repository of such programs or the non-verifiable program is indirectly verifiable by way of a digital signature on the non-verifiable program that proves the program was produced by a trusted source.

I. 35 U.S.C. § 102, Alleged Anticipation, Claims 1-16

Claim 1, which is representative of the other rejected independent claims 11 and 21 with regard to similarly recited subject matter, reads as follows:

1. A method of verifying the integrity of unauthenticated code, comprising:

receiving automatically authenticated code, the automatically authenticated code including an embedded first hash value of the unauthenticated code;

receiving the unauthenticated code;

generating a second hash value of the unauthenticated code;

comparing the first hash value and the second hash value; and

verifying the integrity of the unauthenticated code if the first hash value and the second hash value match.

Appellants respectfully submit that McManis does not teach or suggest the feature of receiving automatically authenticated code, the automatically authenticated code including an embedded first hash value of the unauthenticated code as recited in independent claim 1 of the present invention. Additionally, McManis does not teach that the first hash value is hash value of unauthenticated code. The unauthenticated code is a separate and different entity than the automatically authenticated code.

Claim 1 recites receiving automatically authenticated code, wherein the automatically authenticated code includes an embedded first hash value of the unauthenticated code. A first hash value of unauthenticated code, such as for example native code, is embedded into automatically authenticated code, such as for example JAVA code that references the native code. The automatically authenticated code is received and the unauthenticated code is received; the automatically authenticated code and unauthenticated code are two separate entities. The method to verify the unauthenticated code comprises generating a second hash value from the received unauthenticated code and comparing the generated second hash value of the received unauthenticated code with the embedded first hash value from the automatically authenticated code. As stated previously, this first hash value that is embedded in automatically authenticated code is a hash value of unauthenticated code. McManis does not teach or suggest these features, as they are recited in claim 1.

The Final Office Action states that McManis teaches a program executer receiving ANPrograms (authenticated code), with a hash value within, that a compiling party uses to verify the integrity of each ANProgram as it compiles it into an ASProgram. As shown in **Figure 2**, an ANProgram contains a digital signature of the originating party, which contains a message digest of the ANProgram code (column 5, lines 37-40). In other words, the ANProgram (authenticated code) contains a hash value of itself (authenticated code). To the contrary, the first hash value in

claim 1 of the present application is a hash value of unauthenticated code. In claim 1, a separate unauthenticated code, such as native code called by an automatically authenticated code, is verified by comparing a first hash value of the unauthenticated code embedded in the automatically authenticated code to a second hash value generated from the received unauthenticated code. McManis does not teach or suggest the feature of receiving automatically authenticated code, the automatically authenticated code including an embedded first hash value of the unauthenticated code as recited in claim 1.

The Final Office Action refers to the following portions of McManis in the rejection of claim 1:

> The ANProgram compiler 162 then calls the signature generator 168 and instructs it to generate the ANProgram compiler's digital signature (DigitalSignature$_C$) 320 which can be verified to ensure that the ASProgram 300 was compiled with the trusted ANProgram compiler. This is done in a manner similar to that described earlier for generating the DigitalSignature$_{OP}$. However, in this case, the set of information signed is the ASProgram code and the DigitalSignature$_{OP}$. Another predetermined hash function with a corresponding HashFunction$_C$ ID 324 may be used to generate the message digest MD$_C$ 322 of the set of information to be signed by the DigitalSignature$_C$, the private encryption key of the ANProgram compiler (Compiler's PrivateKey) is used to encrypt the MD$_C$ and the HashFunction$_C$ ID, and the identifier of the ANProgram compiler (Compiler's ID) is added in clear text at the end of the encrypted MD$_C$ and HashFunction$_C$. The Compiler's PrivateKey and ID are provided by the ANProgram compiler.
>
> The ANProgram compiler 162 calls the signature generator 168 a second time to generate the CompParty's digital signature (DigitalSignature$_{CP}$) 312, which can be verified by end users to ensure that the ASProgram 300 was generated by the trusted CompParty. This is done in a similar manner to that described earlier for generating the DigtialSignature$_{OP}$ (in the section discussing preparing an ANProgram for compiling). However, here the message digest (MD$_{CP}$) 314 generated for the DigitalSignature$_{CP}$ is generated by computing a predetermined or selected hash function (HashFunction$_{CP}$) on the data bits of the ASProgram code, the DigitalSignatuer$_{OP}$ and the DigitalSignature$_C$. Similar to the HashFunction$_{OP}$, for purposes of this disclosure, the HashFunction$_{CP}$ corresponds to the CompParty since it was used for the DigitalSignature$_{CP}$ of the CompParty.
>
> The signature generator 168 then encrypts the MD$_{CP}$ 314 and the ID of the HashFunction$_{CP}$ (HashFunction$_{CP}$ ID) 316 with the private encryption key of the CompParty (CompParty's PrivateKey). The signature generator then adds the identifier of the CompParty (CompParty's ID) 318 in clear text at the end of the encrypted items 314 and 316 to form the DigitalSignature$_{CP}$ 312. The CompParty's PrivateKey and ID are provided by the CompParty with the user interface 152.

After the DigitalSignature$_C$ 320 and the DigitalSignature$_{CP}$ 312 are generated, the ANProgram compiler 162 appends them to the ASProgram code 302, so that the resulting compiled ASProgram file or object has the following components in it:

ANProgram Code,
DigitalSignature$_{OP}$,
ASProgram Code,
DigitalSignature$_C$, and
DigitalSignature$_{CP}$.

McManis, column 7, line 25 through column 8, line 11.

> Similarly, the program executer has been described as requiring verification of both a DigitalSignature$_{CP}$ and a DigitalSignature$_C$. However, the program executer could be constructed to require verification of only one of these digital signatures and optionally verify the other digital signature if the ASProgram being verified includes it. Furthermore, the program executer could be constructed to skip the step of verifying the integrity of the ANProgram corresponding to each ASProgram, based on the assumption that the compiling party is trusted and that it is a duty of the compiling party to verify the integrity of each ANProgram that is compiles into an ASProgram prior to performing the compilation.

McManis, column 11, line 61 through column 12, line 6.

These portions of McManis only teach the steps of building a compiled ASProgram. The digital signatures for the compiler and compiling party are created and appended to the ASProgram. The portions describe the contents of the ASProgram as shown in **Figure 3** as follows:

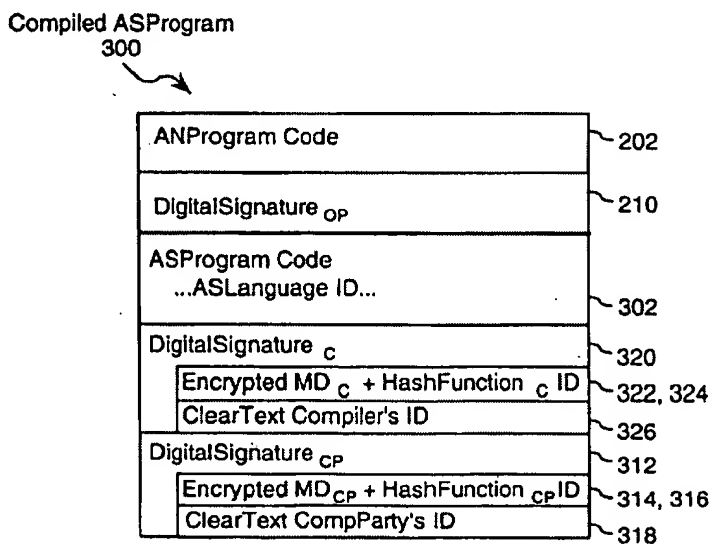

FIG. 3

In the cited portions, McManis teaches authenticating digital signatures of the compiler and compiling party in an unauthenticated program (ASProgram). Figure 3 is a diagram of the compiled program as described in the cited sections. As discussed in the previous response, an ASProgram is unauthenticated code, i.e. the application specific program. McManis does not teach or suggest receiving automatically authenticated code, wherein the automatically authenticated code includes an embedded first hash value of the unauthenticated code as recited in independent claim 1.

In these portions, McManis teaches that a compiled ASProgram is unauthenticated code, which contains ANProgram code, a digital signature of the originating party, ASProgram code, a digital signature of the compiler, and a digital signature of the compiling party. Thus, McManis teaches unauthenticated code may contain a digital signature of automatically authenticated code and a digital signature of unauthenticated code. However, claim 1 recites that automatically authenticated code contains a hash value of unauthenticated code. Since a compiled ASProgram is unauthenticated code, McManis does not teach or suggest the features as recited in claim 1 by describing the contents of a compiled ASProgram.

Cordery is directed toward a method and apparatus for securely transmitting keys from a postage metering apparatus to a remote data center. Cordery teaches a method for transmitting a key from a first device to a remotely located second device includes steps of generating the key within the first device; selecting one of a plurality of one-time pad values from a one-time pad stored within the first device; creating a hash of at least the key and the selected one of the plurality of one-time pad values; and sending the hash and the key from the first device to the second device. Cordery does not teach anything regarding automatically authenticated code or embedding hash values in automatically authenticated code. Additionally Cordery does not teach if the first hash value and the second hash value do not match, receiving the unauthenticated code again, generating a third hash value of the unauthenticated code and comparing the first hash value and the third hash value; wherein if the third hash value and the first hash value do not match, comparing the second hash value and the third hash value, and, if the second hash value and the third hash value match, determining that the unauthenticated code has been corrupted intentionally; and wherein if the second hash value and the third hash value do not match, it is determined that the unauthenticated code has been corrupted unintentionally, as recited in claims

7-9, 17-19 and 27-29. The Office Action alleges that these features are taught by Cordery at column 5, lines 14-22, which reads as follows:

> The data center 222 compares its generated hash value to the received hash value (step S48) and, if they do not match, the data center 222 marks the one-time pad value as used and generates an error message to the postage metering system 202 stating that the newly received public key will be ignored (step S50). The postage metering system 202 user can then either attempt to repeat the process of generating a new key set or request assistance from the postal service.

In this section Cordery merely teaches comparing the generated hash value to the received value and, if they do not match, marks the one-time pad value and generates an error. There is nothing in this section that teaches or suggests the specific features of claims 7-9, 17-19 and 27-29. Moreover, Cordery does not teach or suggest anything regarding comparing a first hash value that is embedded in automatically authenticated code with a second hash value generated from separately received unauthenticated code. Since neither McManis nor Cordery teach or suggest these features, any alleged combination of McManis and Cordery still does not teach or suggest these features.

In response to these arguments, the Advisory Action dated June 7, 2004 states "The applicant argues that the hash value of the unauthenticated code is embedded within the authenticated code, however claim 1 merely state that an embedded hash code of the unauthenticated code is included with the authenticated code." Appellants respectfully disagree. Claim 1, which is representative of the other rejected independent claims 11 and 21 with regard to similarly recited subject matter, recites "the automatically authenticated code including an embedded first hash value of the unauthenticated code." This recitation clearly states that the automatically authenticated code includes an embedded first hash value of the unauthenticated code which means, the automatically authenticated code has a first hash value of the unauthenticated code embedded within it.

Furthermore, there is no teaching or suggestion in either of the references regarding the desirability or even possibility of combining the teachings of McManis with the teachings of Cordery. That is, there is no problem illustrated in McManis for which Cordery teaches the solution, or vice versa. McManis is directed to a system for executing verifiable programs with facility for using non-verifiable programs from trusted sources. Cordery is directed to a system for securely transmitting keys from a postage metering apparatus to a remote data center. There is

nothing in either reference that would suggest to one of ordinary skill in the art that any desired benefit would be obtained from a combination of the distribution systems of McManis with that of Cordery.

Moreover, the alleged motivation provided in the Final Office Action is based completely on a prior knowledge of Appellants' claimed invention. The Final Office Action alleges that the reason one of ordinary skill in the art would be motivated to combine McManis with Cordery is because "determine if the codes has been intentionally or unintentionally corrupted." This alleged motivation is rooted in the erroneous interpretation of Cordery as teaching automatically authenticated code or embedding hash values in automatically authenticated code, as illustrated above. Thus, the allegation that it would be obvious because it would determine if the codes has been intentionally or unintentionally corrupted is not based on the teachings of the references but is instead an attempt to recreate Appellants' claimed invention having first had benefit of Appellants' disclosure. The mere fact that a prior art reference can be readily modified does not make the modification obvious unless the prior art suggested the desirability of the modification. *In re Laskowski*, 871 F.2d 115, 10 U.S.P.Q.2d 1397 (Fed. Cir. 1989) and also *see In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992) and *In re Mills*, 916 F.2d 680, 16 U.S.P.Q.2d 1430 (Fed. Cir. 1993). Moreover, the examiner may not use the claimed invention as an "instruction manual" or "template" to piece together the teachings of the prior art so that the invention is rendered obvious. *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992). Such reliance is an impermissible use of hindsight with the benefit of applicant's disclosure. *Id.*

Additionally, McManis and Cordery, taken alone or in combination, fail to teach or suggest comparing a first hash value that is embedded in automatically authenticated code with a second hash value generated from separately received unauthenticated code. Thus, this is impermissible hindsight reconstruction using Appellants' disclosure as a guide to make the modifications to the references to arrive at Appellants' claimed invention and cannot be used as a proper basis for rejecting the pending claims.

Thus, in view of the above, Appellants respectfully submit that McManis does not teach each and every feature of independent claim 1 or the similarly recited subject matter of independent claims 11 and 21 as required under 35 U.S.C § 102(b). At least by virtue of their dependency on independent claims 1, 11 and 21, the specific features of dependent claims 2-10,

12-20 and 22-29 are not taught or suggested by McManis and Cordery, taken alone or in combination. Accordingly, Appellants respectfully submit that the rejection of claims 1, 2, 4-6, 11, 12, 14-16, 21, 22 and 24-26 under 35 U.S.C. § 102(b) and claims 3, 7-10, 12, 17-20, 23 and 27-29 under 35 U.S.C. § 102(a) should be overturned.

CONCLUSION

In view of the above, Appellants respectfully submit that claims 1-29 are allowable over the cited prior art and that the application is in condition for allowance. Accordingly, Appellants respectfully request the Board of Patent Appeals and Interferences to not sustain the rejections set forth in the Final Office Action.

Respectfully submitted,

Francis Lammes

Francis Lammes
Reg. No. 55,353
YEE & ASSOCIATES, P.C.
PO Box 802333
Dallas, TX 75380
(972) 367-2001

APPENDIX OF CLAIMS

The text of the claims involved in the appeal are:

1. A method of verifying the integrity of unauthenticated code, comprising:

receiving automatically authenticated code, the automatically authenticated code including an embedded first hash value of the unauthenticated code;

receiving the unauthenticated code;

generating a second hash value of the unauthenticated code;

comparing the first hash value and the second hash value; and

verifying the integrity of the unauthenticated code if the first hash value and the second hash value match.

2. The method of claim 1, wherein the automatically authenticated code is compiled platform independent code and wherein the unauthenticated code is native code.

3. The method of claim 1, wherein the automatically authenticated code is a platform independent application or applet and wherein the unauthenticated code is a dynamically linked library.

4. The method of claim 1, wherein the first hash value is obtained using a hashing function and wherein generating a second hash value of the unauthenticated code includes using the same hashing function as was used to obtain the first hash value.

5. The method of claim 4, wherein the hashing function is identified based on information stored in the automatically authenticated code.

6. The method of claim 1, further comprising:

executing the automatically authenticated code using a virtual machine; and

sending a request to a server from which the automatically authenticated code was received, the request being for the unauthenticated code.

7. The method of claim 1, wherein, if the first hash value and the second hash value do not match, the method further comprises:

receiving the unauthenticated code again;

generating a third hash value of the unauthenticated code; and

comparing the first hash value and the third hash value.

8. The method of claim 7, wherein if the third hash value and the first hash value do not match, the method further comprises:

comparing the second hash value and the third hash value; and

if the second hash value and the third hash value match, determining that the unauthenticated code has been corrupted intentionally.

9. The method of claim 8, wherein if the second hash value and the third hash value do not match, it is determined that the unauthenticated code has been corrupted unintentionally.

10. The method of claim 1, wherein the method is implemented in a virtual machine associated with a web browser on a client device.

11. An apparatus for verifying the integrity of unauthenticated code, comprising:

a virtual machine; and

an unauthenticated code verification element, wherein the virtual machine receives automatically authenticated code, the automatically authenticated code including an embedded first hash value of the unauthenticated code, and receives the unauthenticated code, and wherein

the unauthenticated code verification element generates a second hash value of the unauthenticated code, compares the first hash value and the second hash value, and verifies the integrity of the unauthenticated code if the first hash value and the second hash value match.

12. The apparatus of claim 11, wherein the automatically authenticated code is compiled platform independent code and wherein the unauthenticated code is native code.

13. The apparatus of claim 11, wherein the automatically authenticated code is a platform independent application or applet and wherein the unauthenticated code is a dynamically linked library.

14. The apparatus of claim 11, wherein the first hash value is obtained using a hashing function and wherein the unauthenticated code verification element generates a second hash value of the unauthenticated code using the same hashing function as was used to obtain the first hash value.

15. The apparatus of claim 14, wherein the hashing function is identified by the unauthenticated code verification element based on information stored in the automatically authenticated code.

16. The apparatus of claim 11, wherein the virtual machine executes the automatically authenticated code and

sends a request to a server from which the automatically authenticated code was received, the request being for the unauthenticated code.

17. The apparatus of claim 11, wherein, if the first hash value and the second hash value do not match, the virtual machine receives the unauthenticated code again, the unauthenticated code verification element generates a third hash value of the unauthenticated code and compares the first hash value and the third hash value.

18. The apparatus of claim 17, wherein if the third hash value and the first hash value do not match, the unauthenticated code verification element compares the second hash value and the third hash value and, if the second hash value and the third hash value match, determines that the unauthenticated code has been corrupted intentionally.

19. The apparatus of claim 18, wherein if the second hash value and the third hash value do not match, the unauthenticated code verification element determines that the unauthenticated code has been corrupted unintentionally.

20. The apparatus of claim 11, wherein the virtual machine and the unauthenticated code verification element are associated with a web browser on a client device.

21. A computer program product in a computer readable medium for verifying the integrity of unauthenticated code, comprising:

first instructions for receiving automatically authenticated code, the automatically authenticated code including an embedded first hash value of the unauthenticated code;

second instructions for receiving the unauthenticated code;

third instructions for generating a second hash value of the unauthenticated code;

fourth instructions for comparing the first hash value and the second hash value; and

fifth instructions for verifying the integrity of the unauthenticated code if the first hash value and the second hash value match.

22. The computer program product of claim 21, wherein the automatically authenticated code is compiled platform independent code and wherein the unauthenticated code is native code.

23. The computer program product of claim 21, wherein the automatically authenticated code is a platform independent application or applet and wherein the unauthenticated code is a dynamically linked library.

24. The computer program product of claim 21, wherein the first hash value is obtained using a hashing function and wherein the third instructions for generating a second hash value of the unauthenticated code include instructions for using the same hashing function as was used to obtain the first hash value.

25. The computer program product of claim 24, further comprising instructions for identifying the hashing function based on information stored in the automatically authenticated code.

26. The computer program product of claim 21, further comprising:

sixth instructions for executing the automatically authenticated code using a virtual machine; and

seventh instructions for sending a request to a server from which the automatically authenticated code was received, the request being for the unauthenticated code.

27. The computer program product of claim 21, further comprising:

sixth instructions for receiving the unauthenticated code again, if the first hash value and the second hash value do not match;

seventh instructions for generating a third hash value of the unauthenticated code; and

eighth instructions for comparing the first hash value and the third hash value.

28. The computer program product of claim 27, further comprising:

ninth instructions for comparing the second hash value and the third hash value, if the third hash value and the first hash value do not match; and

tenth instructions for determining that the unauthenticated code has been corrupted intentionally, if the second hash value and the third hash value match.

29. The computer program product of claim 28, further comprising eleventh instructions for determining that the unauthenticated code has been corrupted unintentionally, if the second hash value and the third hash value do not match.